International Journal of Smart Home
Vol. 9, No. 12, (2015), pp. 233-244
http://dx.doi.org/10.14257/ijsh.2015.9.12.24

The Improvement and Evaluation of the Implementation Ability for Deriving Timing Constraints Context in Service-Oriented Home Network

BenYan[1], Hua-Ping Yao[1], Masahide Nakamura[2] and Shinsuke Matsumoto[2]

[1] LuoYang Institute of Science and Technology
No. 90 Wangcheng Dadao, Luolong District,
Luoyang City, Henan Province, 471023, China
[2] Kobe University
1-1 Rokkoudait-cho, Nada-ku,Kobe,Hyoko 657-8501,Japan
1 {yanbenjp, yhplisajp}@gmail.com
2 masa-n@cs.kobe-u.ac.jp

Abstract

Home Network System (HNS) is comprised of networked home appliances and sensors to provide value-added and more powerful services. In order to build high-level HNS service by integrating multiple appliances and sensors, our earlier study proposes a method to derive timing constraints context based on Sensor Service Framework (SSF, which deploys sensor devices as web services to achieve easy development of context-aware application).

That method divides timing constraints into two types: the sequential timing constraint and the continuous timing constraint. A high-level context can be defined as conditions expression based on the above two types. Moreover, we present a timer service to implement timing constraints context within SSF, and demonstrate how a high-level context with timing constraint is registered and detected in a real home network system.

However, to create high-level context with timing constraints, the developer needs to know the details about pre-existing context of HNS, and also needs the ability to analyze and implement complex logic to detect a high-level timing constraints context. This limitation impedes the efficient creation of high-level context in HNS.

Therefore, we propose a method to collectively manage the information of a pre-existing context of HNS in this paper, by which the developer can create high-level timing constraints context more easily than before. As a case study, we implement Enter-Leave context and TV Left On context, and execute an evaluation to prove the effectiveness of this proposal.

Keywords*: Home Network System (HNS), service-oriented architecture, context-aware application, web service, Sensor Service Framework (SSF), timing constraints context*

1. Introduction

Home Network System (HNS) is a system consisting of multiple networked household appliances and sensors. It is one of the most promising applications about emerging ubiquitous technologies. The biggest advantage of HNS is that it provides value-added and more powerful HNS services by integrating multiple appliances and various sensors [1]. Recently, the study and development of context-aware application of HNS which can be implemented by matching the surrounding situation or users are becoming more and more notable [6].

A context refers to an information collection which contains the status of user, appliances, surrounding environment and other various entities in HNS. It is defined as a conditional expression by using the value over a single or multiple sensors. The context-aware application means executable software which can be implemented when a context is established.

In our earlier study [8], we proposed a Sensor Service Framework (SSF) based on the idea of service-oriented architecture (SOA) [9]. Since SSF does not depend on any device or platform, it realized loose coupling between applications, appliances and sensors. The developer can use sensor device (such as temperature sensor, illumination sensor etc.) as a standard web service by using standard interfaces in HNS, such as getting value from sensor, registering context condition etc.

We also proposed a Sensor Mush up Platform (SMuP) based on SSF, which is used to build more complex context by combining multiple sensor web services [10], such as the context integrating temperature sensor and moisture sensor [Temperature is 25 degrees or higher, and humidity is 30% or less].

Based on SSF/SMuP, we proposed a method to define and presume a high-level timing constraints context [5]. We divided timing constraints into 2 types: sequential timing constraints and continuous timing constraints. Sequential timing constraints means a relation like context *C1* is detected within *n* seconds after context *C2* is detected. Continuous timing constraints means a relation like context *C* continuously exists during a period of time *n*.

Based on the above proposal, a high-level context can be defined in HNS. Since a new high-level context is combined with one or multiple pre-existing contexts, the developer needs to know the details of these contexts, and needs the ability to analyze and implement complex logic to detect high-level timing constraints context. Thus a problem arises concerning how to use pre-existing contexts to create a new context in context-aware application creating process.

For this purpose, this paper proposes a method to support the developer to create high-level timing constraints context more easily than before. This method includes two web service objects: *ContextRegistry* service and *TimingContextSensor* service.

Context Registry service is used to collectively manage the information of a pre-existing context of HNS. When a developer uses a pre-existing context to create context-aware application, it’s unnecessary to call each sensor service. The details of these sensors and pre-existing contexts can be acquired from *Context Registry* service.

TimingContexSensor service is designed to register a new high-level timing constraints context in HNS, which makes the developing of a context-aware application with a new timing constraints context easier.

To prove the effectiveness of this proposal from performance aspect, we implemented [Enter-Leave context] and [TV Left On] context in our library. We measured the response time of each method and the creation time of a context aware application, which is described in the last part if the paper.

2. Preliminaries

2.1 Home Network System (HNS)

A HNS consists of one or more networked appliances connected to a LAN at home. In general, each appliance has a set of application program interfaces (APIs), by which the users or external software agents can control the appliance via the network. A HNS typically has a home server, which manages all the appliances in the HNS. Services and applications are installed on the home server. A HNS service provides a sophisticated and value-added service by using multiple appliances together. HNS service is implemented as a software application that invokes the APIs of the appliances. The appliances and services are deployed in a home, which is characterized by environmental attributes (e.g., temperature, humidity, brightness, current, sound, space) [2] [3].

In our library, based on service-oriented architecture (SOA) [9], we are developing a HNS texting environment [CS27-HNS] which can use various appliances as web service [11]. Appliances-dependent control method and communication protocol are wrapped by web service, and all appliances in CS27-HNS can be used as a web service of SOAP or REST formality. For example, we can set a TV into 6ch by accessing URL: [*http://cs27-hns/TVService/setChannel?channel=6*].

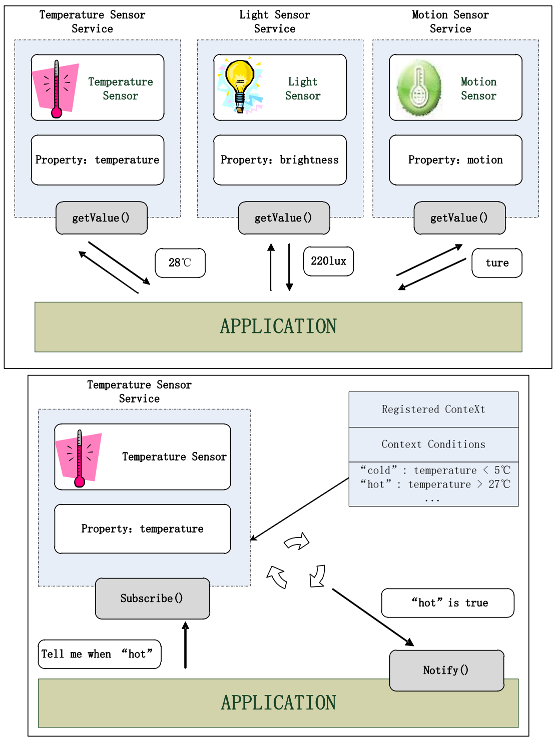

Figure 1. The Summary Figure of Sensor Service Framework (SSF)

2.2 Sensor Service Framework (SSF)

Sensor Service Framework (SSF) was developed as an application framework to deploy sensor devices as web service in CS27-HNS [7] [7] [12]. A web service wraps sensor-specific control logic into standard API. Each sensor has a measurable property. For example, a temperature sensor has a property *temperature (℃)*, a light sensor has a property *brightness(lux)*, and the value of these properties can be acquired form *getValue()* method in CS27-HNS. Moreover, by implementing periodic observation of the change of sensor service's properties, a context can be detected based on the registered conditional expression (Figure 1).

For example, we define a context named "*hot*" and register it as context condition expression: "*temperature > 27*" (join context name and condition together, and we write the above context as [*hot: temperature >27*]). The temperature sensor keeps monitoring "*temperature*" and detects context "*hot*" when the value of "*temperature*" becomes higher than 27℃. The registration of context condition is implemented as "*register()*", and the context can be called by any web service by using "*subscribe()*" method. By using this web service, we can build context-aware service easily just by appointing an appliance of HNS. For example, the HNS service of "*when the room is hot, the air-conditioner will turn on*" can be implemented as [*subscribe(hot,http://cs27-hns/AirConditionerService/on)*]

2.3 Timing Constraints Context

Based on the early study SSF [5], we have presented a method to detect a high-level timing context in HNS. This proposal defined timing constraints context into 2 types: sequential timing constraints context and continuous timing constraints context. Sequential timing constraints context is a sequential time limitation between two contexts, and it means a relation like context *C1* is detected within a few seconds after context *C2* is detected. Continuous timing constraints context is a continuous time limitation that means context *C* continuously exists during a period of time *n*. For example, context [*C: enter home*] can be defined as a sequential timing constraints context with the condition like [*after opening the door for 2 seconds, passed the hall*], and context [*C: appliance is on*] can be defined as a continuous timing constraints context with the condition like [*the power of appliance is ON for 600 seconds*].

3. Research Goal and Approach

3.1 Issues

In our early study, a new high-level context is combined with one or multiple pre-existing contexts. Thus the developer needs to know the details about these HNS contexts, and also needs to have the ability to analyze and implement complex logic to detect high-level timing constraints context [5]. So there is a problem concerning using pre-existing contexts and creating a new context, such as:

- Which sensor service is possible to detect whether an existing context is established or not?
- What conditional expression can judge whether the context is established or not?
- Is the existing context that it itself exists?
- How to implement complex logic between sensor web services and timer web service to measure the time in a new time constrains service?

3.2 Key Idea

To solve the problem, we propose a method to support a developer to create high-level timing constraints context. This method has two main points:

- [K1]: Acquiring details of pre-existing context with calling web service only;
- [K2]: Making the registering of timing constraints context easier.

To achieve K1, we designed *ContextRegistry* service to collectively manage the information of all HNS pre-existing contexts. When a developer uses a pre-existing context to create high-level context, it is unnecessary to call each sensor service registered by the context as before. The details of these pre-existing contexts can be acquired from this web service.

To achieve K2, we implemented *TimingContextSensor* service to register a new high-level timing constraints context easily in HNS.

By these methods, the developing of a context aware application with a new timing constraints context becomes easier. The details of these two services will be described in the next section.

3.3 ContextRegistry Service

ContextRegistry object is designed as a web service to collectively manage the information of all HNS contexts. The information includes the context name, the context condition expression and URL (URL is to get the status of the context as whether established or not). These information can be acquired through the *getContextList()* method in *ContextRegistry* class. A context is established or not can be judged only by calling the *getContextStatus()* method which specifies the context name.

For example, for a context [*T:DoorOpen*] defined as [*DoorOpen: DoorSensor.isOpen == true*], the context establishing status can be acquired by calling "*http://cs27-hns/ContextRegistryService/getContextStatus?name=DoorOpen*" only. It is unnecessary to call each sensor service of the context.

Since a context registers with each sensor service in HNS, the information of sensor service is also managed by *ContextRegistry* class. The information of sensor service includes the name of sensor and URL of sensor service. These information is acquired through the *getSensorList()* method in *ContextRegistry* class.

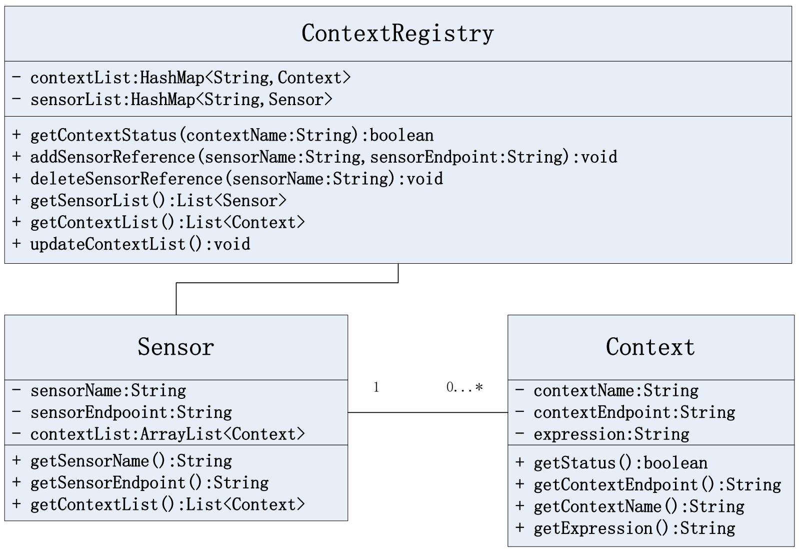
ContextRegistry
- contextList:HashMap<String,Context>
- sensorList:HashMap<String,Sensor>
+ getContextStatus(contextName:String):boolean
+ addSensorReference(sensorName:String,sensorEndpoint:String):void
+ deleteSensorReference(sensorName:String):void
+ getSensorList():List<Sensor>
+ getContextList():List<Context>
+ updateContextList():void
Sensor
- sensorName:String
- sensorEndpooint:String
- contextList:ArrayList<Context>
+ getSensorName():String
+ getSensorEndpoint():String
+ getContextList():List<Context>
1
0...*
Context
- contextName:String
- contextEndpoint:String
- expression:String
+ getStatus():boolean
+ getContextEndpoint():String
+ getContextName():String
+ getExpression():String

Figure 2. Class Diagram of ContextRegistry Service (Class)

Table 1. The Details of Each Class and Method in ContextRegistry Service

Object Name	Object Type	Object Instruction
Sensor	class	This class is to get the name, the URL of a sensor service and the list of the contexts which are registered with the sensors.
Context	class	This class is to get the name, the condition expression and the URL of a context.
getContextStatus	method	This method is to get the context establishing status with specifying the "context name" as an argument.
getSensorList	method	This method is to get the name list and the URL list of the sensor services in HNS.
getContextList	method	This method is to get the name list, the condition expression list and the URL list of the contexts which are registered with sensor services.

Figure 2 is class diagram of the main classes and methods of the *ContextRegistry* service. The details of each class and main methods are indicated in Table 1.

3.4 TimingContextSensor Service

TimingContextSensor object is to register a timing constraints context. Based on the definition of timing constraints context in our earlier study, we design two registration methods to register sequential timing constraints context and continuous timing constraints context.

RegisterSequentialContext() method is to register a new sequential timing constraints context by specifying a new context name (*T*), two pre-existing contexts name (*Cx*, *Cy*) and a time (*n*). It is implemented like *registerSequentialContext*(*T*, *C1*, *C2*,*n*). *T* is the new sequential timing constraints context derived from two pre-existing contexts (*Cx* and *Cy*), which is defined as [*T*: *#n* [*Cx* , *Cy*]]. It means that if a context *Cx* is detected within *n* seconds after context *Cy* is detected, a new context *T* is detected.

R*egisterContinuousContext()* method is to register a new continuous timing constraints context by specifying a new context name (*T*), a pre-existing contexts name (*Cx*) and a time (*n*). It is implemented like *registerSequentialContext*(*T*, *Cx*, *n*). *T* is a new continuous timing context with using pre-context *Cx*, which is defined as [*T*: *@n* [*Cx*]]. It means that if a context *Cx* continuously exists during a period of time *n*, *T* is detected.

S*ubscribe()* method is to connect any web services by appointing the context name and the context calling URL of HNS. The usage is the same as our early study SSF described in chapter 2.2. The same as *ContextRegistry* class (chapter 3.3), a *TimingContextSensor* class includes *getContextList()* and *getSensorList()*, which is to acquire details of the sensor service registered by a context. Figure 3 is the class diagram of the main classes and methods of *TimingContextSensor* service.

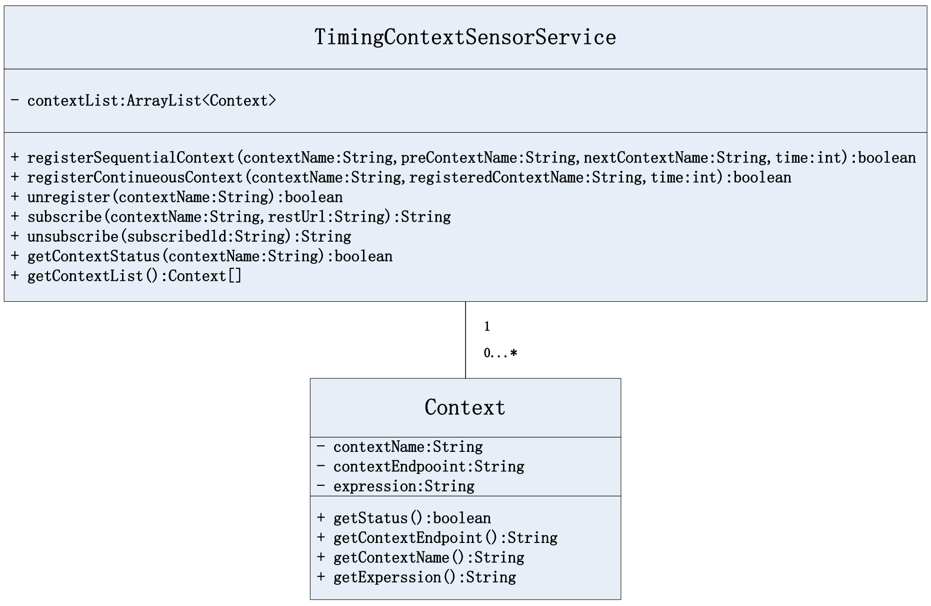
TimingContextSensorService
- contextList:ArrayList<Context>
+ registerSequentialContext(contextName:String,preContextName:String,nextContextName:String,time:int):boolean
+ registerContinueousContext(contextName:String,registeredContextName:String,time:int):boolean
+ unregister(contextName:String):boolean
+ subscribe(contextName:String,restUrl:String):String
+ unsubscribe(subscribedld:String):String
+ getContextStatus(contextName:String):boolean
+ getContextList():Context[]
1
0...*
Context
- contextName:String
- contextEndpooint:String
- expression:String
+ getStatus():boolean
+ getContextEndpoint():String
+ getContextName():String
+ getExperssion():String

Figure 3. Class Diagram of TimingContextSensor Service (Class)

4. Case Study

CS27-HNS is a HNS testing environment in our lab which has deployed *HomeApplianceOperation* services, *TimingContextSensor* service and *ContextRegistry* service [13] [14] [15]. Based on SSF, each sensor of CS27-HNS can be accessed by calling the sensor service. In the case study, we have registered three timing constraints contexts ([*T1*: *Enter*] context, [*T2*: *Leave*] and [*T3*: *TvLeftOn*] context) by using *ContextRegistry* service. This chapter demonstrates how to register these contexts and how to create context-aware applications by using these creating contexts based on our proposal.

4.1 Automatic Light Service (Using Sequential Timing Constrains Context)

This case study creates a context-aware application which combines sequential timing constrains context [*T1*: *Enter*] ([*T2*: *Leave*]) with On/Off service of lights. If somebody enters (or leaves) the room, the light will turn on (or off) by the system automatically.

In CS27-HNS, there are two pre-existing contexts registered as [*C1: DoorOpen*] and [*C2: HumanDetect*]. *C1* is detected by a door sensor at entrance and defined as [*DoorOpen: DoorSensor.isOpen == true*]. *C2* is detected by a motion sensor at entrance and defined as [*HumanDetect: MotionSensor1.motion==true*]. By using the above pre-contexts, a sequential timing constrains context *T1* can be defined as [*Enter*: *#5* [*DoorOpen, HumanDetect*]]. It means if the motion sensor reacts within 5 seconds after the door sensor reacts, somebody enters the room.

The same as *T1*, timing sequential constrains context *T2* can be defined as [*Leave*: *#5* [*HumanDetect, DoorOpen*]]. It means if the door sensor reacts within 5 seconds after the motion sensor reacts, somebody leaves the room.

Based on the proposal in chapter 3, the context *T1* or *T2* can be registered with the *registerSequeintialContext()* method of *TimingContextSensor* service, and the connection between *T1* (or *T2*) and On (Off) service of the lights can be implemented with *subscribe()* method.

The following step1 and step2 indicate the registration calling affiliates of web service for this context-aware application. Step3 indicates the calling method of this context-aware application.

- **STEP1**: Registering Context

  [Registering ***T1***]

  http://cs27-hns/TimingContextSensorService/registerSequeintialContext?name=Enter&context1=DoorOpen&context2=HumanDetect&time=5

  [Registering ***T2***]

  http://cs27-hns/TimingContextSensorService/registerSequeintialContext?name=Leave&context2= HumanDetect&context1= DoorOpen&time=5

- **STEP2**: Creating context-aware application

  [Connecting Light ON with ***T1***]

  http://cs27-hns/TimingContextSensorService/subscribe?context=Enter&notify=http://cs27-hns/LightService/on

  [Connecting Light OFF with ***T2***]

  http://cs27-hns/TimingContextSensorService/subscribe?context=Leave&notify=http://cs27-hns/LightService/off

- **STEP3**: Calling method of context-aware application

  [Power on automatically**]**

http://cs27-hns/TimingContextSensorService/getContextStatus?name=Enter
[Power off automatically)**]**
http://cs27-hns/TimingContextSensorService/getContextStatus?name=Leave

4.2 TV LeftOn Notification Service (Using Continuous Timing Constrains Context)

This section introduces how to create another context-aware application by using pre-existing contexts [*T3: TvLeftOn*], [*C3: NoHumanDetect*], [*C4: Notify*] and [*C5: TvOn*].

T3 is a continuous timing constrains context by using [*C3: NoHumanDetect*] and [*C5: TvOn*]. It is defined as [*TvLeftOn: @600*[[*TvOn*]*&&*[*NoHumanDetect*]]], and means that if the TV is on for 600 seconds and there is nobody in the room, the context will be detected. The new context-aware application is connected with *T3* and *C4*. When *T3* is detected, a notification will be released by this context-aware application automatically.

The same as case study in section 4-1, context *T3* can be registered with the *registerContinuousContext()* method of *TimingContextSensor* service, and the connection between *T3* and *C4* can be implemented with *subscribe()* method.

The following step1 and step2 indicate the registration calling affiliates of web service for this context-aware application. Step3 indicates the calling method of the context-aware application.

- **STEP1:** Register TvLeftOn Context
  http://cs27-
hns/TimingContextSensorService/registerContiueousContext?name=
  TVLeftOn&context5=TVon&context3=NoHumanDetect&time=600
- **STEP2**: Connecting TvLeftOn context with Notify context
  http://cs27-hns/TimingContextSensorService/subscribe?context=TVLeftOn&
  Notify=http://cs27-hns/NotifyService/notify
- **STEP3**: Calling method of context-aware application
  http://cs27-
hns/TimingContextSensorService/getContextStatus?name=TVLeftOn

5. Evaluation

5.1 Evaluation Method and Environment

In order to prove the validity and efficiency of our proposal, we measured the response time and the creating time of each step in context-aware application creating process.

The evaluation is implemented in our HNS testing environment CS27-HNS and the details of evaluation environment and the technique are as follows:

■ Web Server:	Apache Tomcat 7.0
■ Web Service Engineer:	Apache Axis 1.4.1

We first selected 5 items as the response time evaluation objects (Table2 and Table3). Then we called REST 10 times for each URL of the evaluation object and recorded the response time. Finally we took the average of these recorded times as evaluation result (Table2).

For the creating time, we selected TV Left On notification service (context-aware application) as the evaluation object. The creating time of this service includes two parts, one is the time required for getting list of pre-context by using *getContextList()* and registering *TvLeftOn* context by using *registerContinuousContext()* method of the *ContextRegistry* service (the pre-Step and step1 of Table3), the other is the time required to connect this context with notifying service (Step2 of Table3).

5.2 Evaluation Result

Table 2 and table 3 show the evaluation result. For the response time, the total time for registration (Step1) and detection (Step3) of the sequential timing constrains context is longer than the continuous timing constrains context. It's because the sequential timing constrains context connects two contexts, and the continuous timing constrains context only uses one context.

Depending on our proposal, the detailed information of pre-existing context is acquired by calling the *ContextRegistry* service. So we think the response time of any context is the same as this result even including step2 (connecting a context with an operation) of the table2.

Object	Details	evaluation object	response time
Realization of Automatic Light Service (by using timing sequential timing constrains context)			
STEP1	Register Enter Context	■	14.6 milliseconds
STEP2	Connecting Light ON with Enter Context	■	7.2 milliseconds
STEP3	Calling method of context-aware application	■	8.3 milliseconds
Realization of TV Left Notification Service (by using continuous timing constrains context)			
STEP1	Register TvLeftOn Context	■	9.8 milliseconds
STEP2	Connecting TvLeftOn context with Notify context	-	-
STEP3	Calling method of context-aware application	■	12.4 milliseconds

Table 2. The Evaluation Objects for Response Time

For the creating time of context-aware application (TV Left On Notification Service), the steps from getting details of the pre-context (Pre-STEP) to the context registration (step1) cost 67.14 seconds. The step of connecting a context with an operation cost 54.24 seconds. The total creating time is 121.38 seconds.

Object	Details	evaluation object	creating time
Realization of TV Left Notification Service (by using continuous timing constrains context)			
Pre-STEP	Getting details of the pre-context	■	67.14 seconds
STEP1	Register TvLeftOn Context	■	
STEP2	Connecting TvLeftOn context with Notify context	■	54.24 seconds
STEP3	Calling method of context-aware application	-	-

Table 3. The Evaluation Objects for Creating Time

The evaluation result shows that the response time of each method is very short for the application. So we believe that the influence from response performance is small, and that the validity and efficiency of our proposal are enough to develop high-level context-aware application in HNS.

6. Conclusions and Future Work

6.1 Conclusions

In our early study, a new high-level context is combined with one or multiple pre-existing contexts. Thus the developer needs to know the details about these contexts, and needs to have the ability to analyze and implement complex logic to detect high-level timing constraints context.

For this purpose, we present a method to solve the issues described in chapter 3. The goal of this proposal is to assist the developer to create high-level timing constraints context much more easily.

our proposal includes two main web services, *ContextRegistry* service and *TimingContextSensor* service. *ContextRegistry* service is to collectively manage the information of a pre-existing context of HNS. *TimingContextSensor* service is to register new timing constraints context and create the context-aware application by using these contexts.

ContextRegistry service includes three main methods: *getContextList*() method, *getContextStatus()* method and *getSensorList*() method. G*etContextList()* method is to get the details of pre-existing context, such as context name, context condition expression and URL for acquiring context status. G*etContextStatus ()* method is to get the status to indicate a context is established or not. G*etSensorList ()* method is to get the details of a sensor service which was registered by a context.

Considering the special nature of timing constraints context, we also proposed *TimingContextSensor* service that includes a *registerSequentialContext*() method and a *registerContinuousContext*() method. Both methods are used to register timing constraints context more easily in HNS.

As a case study, we created two context-aware applications using the pre-existing timing constraints contexts in our lab. One is the automatic light service used to switch on or off the lights when somebody enters or leaves the room. Another is TV left on notification service which gives a notification when the TV is on for 600 seconds with nobody in the room. Moreover, we measured the response time of each method in the *TimingContextSensor* service and the creation time of the above context-aware applications. The evaluation result proved the effectiveness of this proposal from performance aspect.

6.2 Future Work

Although the creating of context-aware application became easier than before depending on the above proposal, the condition of each context itself becomes more and more complex. It's difficult to inspect conflict between contexts beforehand. Because of this, we plan to develop a framework in order to inspect conflict between pre-contexts beforehand in context-aware application creating process in our future research。

Furthermore, even the creating process is easier than before, it still needs several minutes to complete the whole steps. If a developer is unaccustomed to the developing process, it is possible to cause some mistakes in the context-aware application. Thus, another future work is the improvement plan to develop a GUI to support context-aware application development.

Acknowledgements

This research was partially supported by the Scientific Research Start Funds of LuoYang Institute of Science and Technology (No.14308051); the Japan Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for Scientific Research (B) (No.26280115, No.15H02701), Young Scientists (B) (No.26730155)]; and the International scientific and technological cooperation project (2015) supported by Science and Technology Department of Henan Province, China.

References

[1] B. Yan, M. Nakamura, L. D. Bousquet and K. I. Matsumoto, "Validating Safety for Integrated Services of Home Network System Using JML", Journal of Information Processing (JIP), vol. 49, no. 6, **(2008)**, pp. 1751-1762.

[2] B. Yan, M. Nakamura and K. I. Matsumoto, "Deriving Safety Properties for Home Network System Based on Goal-Oriented Hazard Analysis Model", International Journal of Smart Home, vol. 3, no. 1, **(2009)**, pp. 67-80.

[3] B. Yan, M. Nakamura, L. D. Bousquet and K. I. Matsumoto, "Improving Reusability of Hazard Analysis Model with Hazard Template for Deriving Safety Properties of Home Network System", International

Journal of Smart Home, vol. 3, no. 2, **(2009)**, pp. 71-88.

[4] L. D. Bousquet, M. Nakamura, B. Yan and H. Igaki, "Using Formal Methods to Increase Confidence in a Home Network System Implementation: a Case Study", Innovations in Systems and Software Engineering (ISSE Journal), vol. 5, no. 3, **(2009)**, pp. 181-196.

[5] B. Yan, H. P. Yao, M. Nakamura and S. Matsumoto, "A proposal for deriving timing constraint context on using multiple sensor web servers in Service-Oriented Home Network", International Journal of Smart Home, vol. 9, no. 8, **(2015)**.

[6] B. N. Schilit, N. Adams and R. Want, "Context-Aware Computing Applications", Proceedings of the 1st IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), **(1994)**; Washington, DC, USA.

[7] A. K. Dey and G. D. Abowd, "Towards a Better Understanding of context and context-awareness", Proceesing of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC), **(1999)**; Karlsruhe, Germany.

[8] H. Sakamoto, H. Igaki and M. Nakamura, "A Sensor Service Framework for Context-aware Applications", Technical Report of the Institute of Electronics, Information and Communication Engineers, vol. 108, no. 458, **(2009)**, pp. 381-386.

[9] T. Erl, "Service-Oriented Architecture: Concepts, Technology and Design", Prentice Hall, **(2008)**.

[10] H. Sakamoto, H. Igaki and M. Nakamura, "SMuP:A Service-oriented Platform for Sensor Service Mashups", Winter workshop in Kurasiki, vol. 2010, no. 3, **(2010)**, pp. 73-74.

[11] A. Tanaka, M. Nakamura, H. Igaki and K. I. Matsumoto, "Adapting Conventional Home Appliances to Home Network Systems Using Web Services", Technical Report of the Institute of Electronics, Information and Communication Engineers, vol. 105, no. 628, **(2006)**, pp. 67-72.

[12] M. Fukuda, H. Seto, H. Sakamoto, H. Igaki and M. Nakamura, "A Looking Back Service for Power Consumption Logs in Home Network System", Technical Report of the Institute of Electronics, Information and Communication Engineers, vol. 109, no. 272, **(2009)**, pp. 29-34.

[13] M. Nakamura, H. Igaki, H. Tamada and K. I. Matsumoto, "Implementing Integrated Services of Networked Home Appliances Using Service-oriented Architecture", the Journal of Information Processing Society of Japan, vol. 46, no. 2, **(2015)**, pp. 314-326.

[14] Phidgets Inc. Unique and Easy to Use USB Interface, http://www.phidgets.com/.

[15] S. Matsuo, H. Seto, H. Sakamoto, H. Igaki and M. Nakamura, "Sensor search with spatial information and support by showing similar parameter for building sensor context", Technical Report of the Institute of Electronics, Information and Communication Engineers, vol. 109, no. 327, **(2009)**, pp. 59-64.

Authors

Ben Yan, he received the B.E. degree in Henan University of Science and Technology, China, in 1999, M.E. degree in Department of Information Science Okayama University of Science, Japan, in 2006, and Ph.D. degree in the Graduate School of Information Science at Nara Institute of Science and Technology, Japan, in 2008. From 2009 to 2014, he worked for Panasonic Group, SANYO Information Technology Solutions Co., Ltd, Osaka, Japan. He is currently a professor in the Department of Computer and Information Engineering at Luoyang Institute of Science and Technology (LIT). His main research interests include the service-oriented architecture, the V&V of home network systems, and requirements engineering for safety critical systems.

HuaPing Yao, she received the B.E. degree in Henan University of Science and Technology, China, in 1999, M.E. degree in Department of Information Science Okayama University of Science, Japan, in 2006. From 2006 to 2014, he worked for CSI and Trend Creates Co., Ltd, Osaka, Japan. She is currently a lecturer in the Department of Computer and Information Engineering at Luoyang Institute of Science and Technology (LIT). Her main research interests include the e-learning, software engineering and home network system.

Masahide Nakamura, he received the B.E., M.E., and Ph.D. degrees in Information and Computer Sciences from Osaka University, Japan, in 1994, 1996, 1999, respectively. From 1999 to 2000, he has been a post-doctoral fellow in SITE at University of Ottawa, Canada. He joined Cyber media Center at Osaka University from 2000 to 2002. From 2002 to 2007, he worked for the Graduate School of Information Science at Nara Institute of Science and Technology, Japan. He is currently an associate professor in the Graduate School of System Informatics at Kobe University. His research interests include the service /cloud computing, smart home, smart city, and life log. He is a member of the IEEE, IEICE and IPSJ.

Shinsuke Matsumoto, he received the B.E. degree in computer science from Kyoto Sangyo University in 2006. He received M.E. and Ph.D. degrees in information science from Nara Institute of Science and Technology in 2008 and 2010, respectively. He is currently an assistant professor in the Graduate School of System Informatics at Kobe University. His research interests include software engineering, mining software repository and cloud computing.